Congestion-Oriented Shortest Multipath Routing

Shree Murthy **J.J. Garcia-Luna-Aceves**
Computer Engineering Department
University of California
Santa Cruz, CA 95064

Abstract

We present a framework for the modeling of multipath routing in connectionless networks that dynamically adapt to network congestion. The basic routing protocol uses a short-term metric based on hop-by-hop credits to reduce congestion over a given link, and a long-term metric based on end-to-end path delay to reduce delays from a source to a given destination. A worst-case bound on the end-to-end path delay is derived under three architectural assumptions: each router adopts weighted fair queueing (or packetized generalized processor sharing) service discipline on a per destination basis, a permit-bucket filter is used at each router to regulate traffic flow on a per destination basis, and all paths are loop free. The shortest multipath routing protocol regulates the parameters of the destination-oriented permit buckets and guarantees that all portions of a multipath are loop free.

1. Introduction

Efficient routing results in smaller average packet delays, which means that the flow control algorithm can accept more traffic into the network. On the other hand, an efficient flow control algorithm rejects excessive offered load that would necessarily increase packet delays by saturating network resources. It is clear that routing and congestion control are very much interrelated.

A drawback of existing internet routing protocols is that their main routing mechanisms (route computation and packet forwarding) are poorly integrated with congestion control mechanisms. More specifically, today's internet routing is based on single-path routing algorithms; even in theory, a routing protocol based on single-path routing is ill suited to cope with congestion, because the only thing the protocol can do to react to congestion is changing the route used to reach a destination. However, as has been documented in [1], allowing a single-path routing algorithm to react to congestion can lead to unstable oscillatory behavior. Furthermore, for connectionless service, any datagram offered to the network is accepted; although routers forward packets only on a best-effort basis and drop them when congestion occurs, the steps taken by routers occur after the packets have been allowed to congest the network, and it is up to the transport protocol to react to congestion *after* network resources are already being wasted.

The work reported in this paper was motivated by our conjecture that architectural elements similar to those used in a connection-oriented architecture to allow the network to enforce performance guarantees could be used to integrate routing with congestion control, and to provide some delay guarantees for the delivery of those datagrams that are accepted in the network. We propose a new framework and protocol for dynamic multipath routing in packet-switched networks that attempts to prevent over utilization of network resources and hence congestion. Packets are individually routed towards their destinations on a hop by hop basis. A packet intended for a given destination is allowed to enter the network if and only if there is at least one path of routers with enough resources to ensure its delivery within a finite time. In contrast to existing connectionless routing schemes, once a packet is accepted into the network, it is delivered to its destination, unless resource failures prevent it. Each router reserves buffer space for each destination, rather than for each source-destination session as it is customary in a connection-oriented architecture, and forwards a received packet along one of multiple loop-free paths towards the destination. The buffer space and available paths for each destination are updated to adapt to congestion and topology changes.

Our framework is based on three main architectural elements: (a) traffic shaping by means of destination-oriented permit buckets; (b) traffic separation and scheduling on a per destination basis; and (c) the dynamic maintenance of multiple loop-free paths that always attempt to reduce the delay from source to destination. Permit buckets consist of permits or tokens fed by periodic updates of credits. To schedule packet transmission, we assume a packet-by-packet generalized processor sharing (PGPS) server [9] at each node. To establish loop-free multipaths, we extend prior results on loop-free single-path routing algorithms introduced in [5]. This results in a congestion-oriented multipath routing architecture that uses a short-term metric based on hop-by-hop credits to reduce congestion over a given link, and a long-term metric based on end-to-end path delay to reduce delay from source to destination. The main contribution of this work is to illustrate the provision of performance guarantees in a connectionless routing architecture.

Section 2. describes the network model used in our protocol. Section 3. gives a detailed description of the new routing protocol. Section 4. derives worst-case steady-state delay bounds for packets accepted into the network by extending the analysis described in [10]. Section 5. presents our conclusions.

2. Network Model

A computer network is modeled as an undirected finite graph represented by $G(N, E)$, where N is the set of nodes and E is the set of edges or links connecting the nodes. A functional bidirectional link connecting nodes i and j is represented as (i, j) and is assigned a positive weight in each direction. A link is assumed to exist in both directions at the same time. All routing messages that are received (transmitted) by a node are put in the input (output) queue on a *FCFS* basis and are processed in that order. Each node is represented by a unique identifier and the link costs can vary in time but are always positive. The distance between any two given nodes is measured as the sum of the link costs of the path between the nodes.

A *path* from node i to node j is a sequence of nodes, $i, n_1, ..., n_r, n_j$, where (i, n_1), (n_x, n_{x+1}), (n_r, j) are links in the path. A *simple path* from i to j is a sequence of nodes in which no node is visited more than once. A *multipath* from i to j

* This work was supported in part by the Office of Naval Research under Contract No. N-00014-92-J-1807 and by the Advanced Research Projects Agency (ARPA) under contract F19628-93-C-0175

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1996		**00-00-1996 to 00-00-1996**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Congestion-Oriented Shortest Multipath Routing	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**9**	

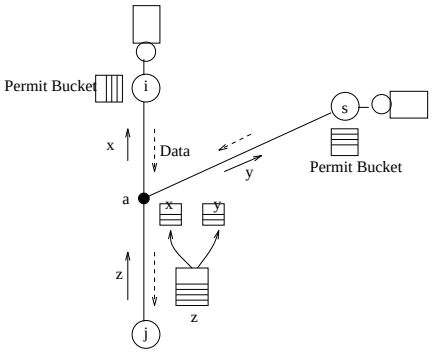

Fig. 1. Credit Aggregation

is a set of simple paths from i to j. The paths between any pair of nodes and their corresponding distances change over time in a dynamic network. At any point in time, node i is connected to node j if a physical path exists from i to j at that time. The network is said to be connected if every pair of operational nodes are connected at a given time.

3. Protocol Description

The new protocol can be divided into three functional elements, namely: *packet scheduling and transmission*, *congestion-based credit mechanism* and *maintenance of multiple loop-free paths*.

Scheduling at a node is done by maintaining permit bucket filters at each node for all active destinations. A weighted fair queueing mechanism is used for fairness [4]. Routing is done on a hop-by-hop basis independently at each node.

To forward packets to a given destination, the protocol uses two routing metrics: a short-term metric based on hop-by-hop credits to reduce congestion along a link, and a long-term metric based on path-delay to minimize end-to-end delay.

The routing variables associated with each link is determined by periodically monitoring traffic on the incoming and the outgoing links at each node through each neighbor. Given the capacity of each link, the utilization of the link can also be determined. Credits are reassigned to upstream neighbors depending on the traffic flow on each of the incoming links. A multipath routing algorithm based on DUAL [5] maintains multiple loop-free paths. Each time the network state changes, paths are recomputed and the new network state is obtained. This is made possible by the periodic exchange of routing information.

Each node maintains a *routing table*, a *distance table*, a *link cost table* and a *link credit table*. The distance table at node i is a matrix that contains, for each destination j and for each neighbor k, routing (cost and credit) information along with the distance (D^i_{jk}) reported to node i by node k regarding destination j, and a successor flag ($flag^i_{jk}$) indicating whether neighbor k belongs to the shortest multipath set, for destination j. Node i's routing table is a column vector containing the routing information about the shortest path to all destinations; it maintains information about the distance (D^i_j), successor (s^i_j), and the routing parameters (credits and delay). The neighbor nodes used for packet forwarding from node i to node j are said to belong to the *shortest multipath* from i to j, denoted by SM^i_j. If the neighbor node belongs to SM^i_j, then $flag^i_{jk}$ is set to 1; otherwise it is set to 0. The link cost table maintains the distance information about all the neighboring links and the link-credit table maintains information about the credits available through all the neighboring links for each destination.

3.1 Packet Scheduling and Transmission Scheme

Packet scheduling is done by means of permit bucket filters for each destination. The *packet-by-packet generalized processor sharing* (PGPS) scheme is used at each server [9]. Packets are transmitted as individual entities. A packet is said to have arrived only after the last bit has been received at a node. The server picks up the first packet that would complete service if no additional packets would arrive. Routing is done on a per destination basis over multiple paths.

Note that, because all the nodes along any path from a source to a given destination can contribute to the flow to that destination, each node is modeled as a PGPS server to regulate the incoming traffic, instead of just having a simple scheduling discipline at the intermediate nodes as can be assumed in a connection-oriented architecture [10]. This scheme, along with the credit-based congestion control mechanism, ensures that the bursty nature of sources does not affect the routing architecture.

The protocol uses two routing metrics for transmitting packets to a given destination: a short-term metric based on hop-by-hop credits to reduce congestion along a link, and a long-term metric based on path-delay to minimize end-to-end delay along the paths. The number of packets sent to a neighbor depends on the credits available through that neighbor. Credits for a destination are sent from a destination towards the source along the reverse paths implied by the routing tables. When a node becomes operational, depending on the availability of resources at each node, credits are distributed among its neighboring nodes.

The traffic at each node is regulated by permit-buckets, independently for each destination. In the traditional leaky bucket congestion control scheme, buckets are session oriented. Data packets accepted from the transmitter and the average rate of flow is controlled by a burst rate for a source-destination session. In our scheme, permit buckets (which are similar to leaky buckets) are destination oriented. For a given destination j, credits arrive to a given node i at a rate ρ^i_j, which is called the *token generation rate* for destination j at node i. The bucket size, denoted by $\sigma^i_j(t)$ gives the maximum number of packets that can be transmitted from i to j at time t. This determines the burstiness of traffic, and $\sigma^i_j(t)$ is defined for each destination j at time $t \geq 0$ as

$$\sigma^i_j(t) = l^i_j(t) + Q^i_j(t) \tag{1}$$

where $l^i_j(t)$ is the number of left-over credits (or tokens) in the bucket at node i for destination j at time t, and $Q^i_j(t)$ is the backlog for destination j at time t. This definition is much the same given in [9], the only difference being that here we maintain leaky-bucket parameters for each active destination rather than for each session.

Destination-based credits are aggregated at each node. Each hop is considered as a source; credits sent by the downstream nodes are aggregated at each hop for a given destination and are redistributed among its upstream neighbors. The total available credits at each node for a given destination is the sum of the credits received from its downstream neighbors for that destination. In Figure 1, if z is the number of credits received by node a from its downstream neighbors to destination j, then node a maintains a permit bucket of size z. These credits are redistributed among its upstream neighbors depending on the traffic flow along links (i, a) and (s, a) as x and y.

The number of credits left behind denoted by l^i_j, is the difference in the number of arrivals and the number of credits that arrive within a given time interval. Accordingly, $l^i_j(\tau, t) = A^i_j(\tau, t) - [K^i_j(t) - K^i_j(\tau)]$, where $K^i_j(t)$ is the number of credits

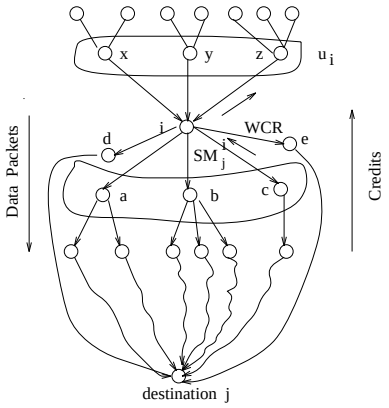

Fig. 2. Multipath Tree

that arrive at node i at time t for destination j and $A_j^i(\tau, t)$ is the traffic arriving at node i for destination j in the interval $(\tau, t]$. The total number of accepted credits in a time period should be less than the credit generation rate. Therefore, with $\tau \leq t$,

$$K_j^i(t) - K_j^i(\tau) \leq \rho_j^i(t - \tau) \tag{2}$$

and

$$\sigma_j^i(\tau, t) \geq A_j^i(\tau, t) - \rho_j^i(t - \tau) + Q_j^i(\tau, t) \tag{3}$$

For the time interval $(t - \Delta t, t)$ we can write Eq. 3 as:

$$\sigma_j^i(t) \geq A_j^i(t) - \rho_j^i(t) + Q_j^i(t) \tag{4}$$

ρ_j^i is related to the number of total available credits at time t and is the sum of the credits available through all the nodes downstream of node i. Consider Figure 2, the total number of credits available at router i for destination j is the sum of the credits available from its downstream neighbors a, b, and c for destination j. The total number of packets transmitted to destination j from node i cannot exceed the total available credits at i for j at time t. The number of available credits also depends on the traffic flow on that link which is a measure of the congestion level of that link.

3.2 Credit-Based Congestion Mechanism

Congestion over a given link is controlled by a hop-by-hop credit-based mechanism. Each node selects a path to a destination based on the bandwidth available through a given link, utilization of that link, and the distance to the destination. The chosen path is subjected to a constraint that the bandwidth available is at least equal to the required bandwidth, and the total bandwidth allocated through a link is less than the capacity of that link. The available bandwidth is then translated into credits. A credit given by a node to its upstream neighbors for a given destination represents the number of packets that a node can accept from its neighbors for a destination. Credits are sent to upstream nodes along the specified reverse direction of the routing table.

Figure 3 presents a formal description of the allocation scheme. Procedure *Initialize* indicates the action taken by a node when it becomes active for the first time. Procedure *Receive* describes the functions performed by a node when a node receives a periodic update.

3.2.1 Initialization

The number of credits available at each node is determined by the buffer space available at that node ($MaxBufs_j$). A part of the total available credits ($Reserve$) is reserved for fast reservation mechanism. A fast reservation mechanism is used to allocate credits to a new node when it becomes a part of the loop-free path to a given destination. This mechanism sends a minimum number of credits to the new shortest multipath neighbor as explained below. This speeds up the credit allocation process, thus avoiding slow start. The remaining credits, CR_j^i, are equally distributed among the neighboring nodes, N_i, on startup. This is termed as the weighted credit WCR_{jk}^i. The delay incurred for the credits to reach neighbors is incorporated while computing the available credits.

On initialization, credits are equally distributed among neighbors since there will not be any traffic on any of the links of a newly established node. Credits are dynamically assigned thereafter among all the active flows, depending on the traffic flow through each of the links. When an operational node i recognizes that a new neighbor has become operational, it sends a fixed minimum number of credits (CR_{min}) to its new neighbor indicating that i can be a possible multipath successor. When node i selects neighbor k as one of its multiple successors to a destination, it sets the successor flag in the update message ($flag_{jk}^i$) to that neighbor to indicate that the neighbor now belongs to SM_j^i. When node k recognizes this, it includes the node in its set of active neighbors and sends minimum credits to the neighbor and redistributes its credits for that destination. This information is communicated to other nodes in the next update interval. The total credits sent to the upstream neighbor is limited by the total available credits at that node. Credit information at each node is updated periodically.

Each routing node resets its traffic counters and monitors the incoming and outgoing traffic for all its neighbors. Based on this statistics, the routing parameters ϕ_{jk}^i are computed. The permit bucket parameters are also initialized for each destination. The token generation rate ρ_j^i is initialized to the sum of the credits available through all the neighbors of i to a given destination j in the given time period. The bucket size σ_j^i is initialized to the number of leftover packets since on initialization there will not be any backlog.

3.2.2 Steady State

A periodic update timer is maintained at each router to exchange credit information periodically. Each router monitors its traffic on its incoming and outgoing links every Δt seconds (update interval). The update interval Δt should be longer than the maximum round trip time (RTT) delay between two nodes in the network. Each time an update is sent, the timer, $timer_j^i$, is reset (Figure 3).

At each node, credits received from all downstream nodes are aggregated and are redistributed to the upstream neighbors. This can be done because the total bandwidth allocated at each link at any given time is no more than the capacity of that link. A node can send data packets to a downstream neighbor only if the credit value through that neighbor is greater than zero. Also, because at each hop credits are distributed based on the traffic flow, the algorithm ensures that information about active destinations is maintained, i.e., those for which data traffic needs to flow from or through the node. When a new destination for which the bandwidth is not reserved becomes active, or when a node becomes a part of the set of loop-free paths to a destination, credits are redistributed using a fast reservation mechanism.

Each node monitors the traffic flowing through it periodically and determines the traffic flow on each of its links for all destinations. It also computes the end-to-end delay associated with

Variables:

p^i_{jl}: credits occupied by packets in transit
q^i_j: credits due to packets already in queue

Procedure Initialize
when router i initializes itself
begin
$CR^i_j \leftarrow MaxBufs_j - Reserve$
do for $k \in N_i$
begin
$WCR^i_{jk} \leftarrow \frac{CR^i_j}{|N_i|}$
$flag^i_{jk} \leftarrow 0$
end
Send credit information to all $x \in N_i$ at next update interval
Reset $timer^i_j$
end

Procedure Receive(k)
when a periodic update is received ($timer^i_j$ expired)
begin
if $((flag^i_{jk} = 1) \wedge (k \notin SM^i_j))$
$WCR^i_{jk} \leftarrow CR_{min}; SM^i_j \leftarrow SM^i_j \cup k$
if $((flag^i_{jk} = 0) \wedge (k \in SM^i_j))$
$WCR^i_{jk} \leftarrow 0; SM^i_j \leftarrow SM^i_j - k$
$\forall m \in N_i$ **begin**
if $m \in SM^i_j$ $flag^i_{jm} \leftarrow 1$
else $flag^i_{jm} \leftarrow 0$
end
redistribute credits among shortest path neighbors
$CR^{i'}_j \leftarrow \sum_{l \in SM^i_j} WCR^i_{jl} - p^i_{jl}$
$CR^i_j \leftarrow CR^{i'}_j - q^i_j$
$WCR^i_{jk} \leftarrow CR^i_j \times \phi^i_{jk} \mid i \in SM^k_j$
Send credit information to all $x \in N_i$ at next update interval
Reset $timer^i_j$
end

Fig. 3. Credit Distribution Mechanism

packets to each destination. If the measured delay does not satisfy the required QoS, that successor will no longer be selected as a feasible successor to that destination and this information is communicated to all the neighboring nodes. It then determines the total available credits for a given destination j and the credits are redistributed among its upstream neighbors after reserving a fraction of the credits for the initialization phase. The philosophy behind this mechanism is similar to a fast bandwidth reservation scheme in which, the data transmission begins before a connection has been completely established.

Figure 4 shows the distance table at node i for destination j for the configuration in Figure 2. The flag field indicates whether the neighbor belongs to the shortest multipath set or not. The distance gives the sum of the link costs along the path to destination j and credits gives the number of available credits through that path. A credit of 0 implies that packets cannot be forwarded through that path.

The number of credits available at a node is determined by the flow on its links and the total traffic seen by a node. If f^k_{ji} is the incoming flow on link (k,i) to destination j as seen by node i, and r^i_j is the traffic originated at i for destination j, we define the total input traffic seen by i for destination j as the sum of all the incoming traffic at node i, and denote it by t^i_j. Furthermore, by the conservation of flow, the sum of all the traffic arriving at a node must be equal to the sum of all the traffic departing from a node for each destination j. Therefore, for destination j, the total incoming flow is equal to the total outgoing flow at node i, and

$$t^i_j = \sum_{k \mid i \in SM^k_j} [f^k_{ji}] + r^i_j = \sum_{m \in SM^i_j} f^i_{jm} \tag{5}$$

For convenience, a routing variable, denoted by ϕ^i_{jk}, is defined for each link (i,k) as the ratio of the flow on each link with respect

Destination	Neighbor	Flag	Distance	Credits
j	a	1	a1	a2
j	b	1	b1	b2
j	c	1	c1	c2
j	d	0	d1	0
j	e	0	e1	0
j	x	0	x1	0
j	y	0	y1	0
j	z	0	z1	0

Fig. 4. Distance Table at node i for destination j

to the total flow on all outgoing links for a given destination j. From Eq. 5 and with N_i denoting the neighbor set of i, we have:

$$\phi^i_{jk} = \frac{f^i_{jk}}{t^i_j} \ \forall k \in N_i \tag{6}$$

Because node i itself can also contribute to the total traffic, by the conservation of flow, it must be true that

$$\sum_{k \in SM^i_j} \phi^i_{jk} \ \leq \ 1 \tag{7}$$

The distribution of credits to upstream neighbors depends on the traffic flow on that link, which in turn depends on the routing variable ϕ^i_j associated with that link. The number of credits a node sends to an upstream neighbor is called the weighted credit (WCR). Credits are weighted by the traffic flow on a given link. The token generation rate for a given update period Δt can now be defined as

$$\rho^i_j(t) = \frac{Credits \ \ available \ in \ update \ period \ before \ t}{\Delta t}$$

$$\rho^i_j(t) = \frac{\sum_{k=1}^{SM^i_j} WCR^i_{jk}(t)}{\Delta t} \tag{8}$$

To obtain a correct estimate of the credits available at each node at any given time, we need to take into account the delay associated with the propagation of credits. This can be done either by estimating the credits available as in [6] or by explicitly sending a marker. We opt for the estimation mechanism. Here, credits are sent to the immediate upstream neighbor, i.e., they propagate only one hop. The update period used for updating routing information is considered as one round-trip delay by a data packet. Therefore, to obtain a correct estimate of the available credits at a node, we have to take into account the data packets that the sender has already forwarded over the link for the past round-trip time (RTT) and the data packets that are already queued from the past RTT. Therefore, the total available credits at node i for a destination j, denoted by CR^i_j, is the difference between the sum of all the weighted credits available from its downstream neighbors d_i (equivalently, sum of all the credits on its outgoing links) belonging to the shortest multipath and the credits which are already being used, i.e.,

$$CR^i_j = \sum_{l \in SM^i_j} [WCR^i_{jl} - p^i_{jl}] - q^i_j \tag{9}$$

where WCR^i_{jl} is the weighted credit obtained from the downstream neighbor l over an update period (which depends on the flow on the link (i,l)), p^i_{jl} is the number of credits occupied by the packets that are already in transit on link (i,l), and q^i_j is the number of credits due to the data packets that are already in the queue at node i for destination j which were not completely transmitted since the previous update period. If a node does not have credits for a given destination j, then CR^i_j is set to zero.

The correctness of the credit based mechanism (i.e., showing that it has no deadlocks and that packets are not dropped) can be proven in a similar way as for virtual-circuit connections [8]. For the purposes of such a proof, it must be assumed that initialization of the protocol is done properly, that there are no link errors, and that there are no link and node failures.

3.3 Maintenance of Loop-Free Multipaths

The primary objective of maintaining multiple loop-free paths is to minimize the end-to-end path delay by reducing network congestion along the path. The distance reported by neighbor k to node i for destination j is denoted by D^i_{jk} and node i's distance to its neighbor k is denoted by d_{ik}. The distance to neighbor k is the sum of the propagation delay δ^i_k and the per hop packet delay through neighbor k, d^i_{jk}. i.e., $d_{ik} = d^i_{jk} + \delta^i_k$. The path delay at node i along node k at a given time t, denoted by $\check{D}^i_{jk}$ is

$$\check{D}^i_{jk} = D^i_{jk} + d_{ik}$$

The *shortest multipath set* of i for destination j (SM^i_j) are those neighbors of i that provide loop-free paths to j. The delay at node i to destination j at time t is computed as the weighted average path delay through all the nodes in the shortest multipath at node i; it is denoted by $D^i_j(t)$. This delay is weighted by the fraction of the traffic going through that path, i.e.,

$$D^i_j(t) = \sum_{k \in SM^i_j(t)} \phi^i_{jk}(t).\check{D}^i_{jk}(t) = \sum_{k \in SM^i_j(t)} \frac{f^i_{jk}(t)}{t^i_j(t)}.\check{D}^i_{jk}(t) \tag{10}$$

The flow from i to each neighbor in SM^i_j depends on the credits available through that neighbor. Assuming that packets are of fixed size and that each packet corresponds to one credit, we can say that a packet flows on a link if at least one credit is available on that link. This implies that the number of packets that can flow on a link is equal to the number of credits available through that link; therefore,

$$D^i_j(t) = \frac{1}{t^i_j(t)} \sum_{k \in SM^i_j(t)} [WCR^i_{jk}(t).\check{D}^i_{jk}(t)] \tag{11}$$

If the packets are of variable lengths, the packet length is a multiple of credits and for simplicity we can assume that each packet requires C credits on an average. The total number of packets transmitted along a link with WCR^i_{jk} credits is a constant K times the total available credits; therefore,

$$D^i_j(t) = \frac{K}{t^i_j(t)}. \sum_{l \in SM^i_j(t)} WCR^i_{jl}(t).\check{D}^i_{jl}(t) \tag{12}$$

Multiple loop-free paths from each node to a destination are maintained by means of a shortest multipath routing algorithm (SMRA), which is based on DUAL [5]. Any change in distance is notified by event-driven update messages. An update message from router i consists of a vector of entries; each entry specifies a destination j, an update flag, a successor flag, the reported distance to that destination and the reported credits available to destination j through that neighbor. The update flag indicates whether the entry is an update ($u^i_j = 0$), a query ($u^i_j = 1$) or a reply to a query ($u^i_j = 2$).

A detailed specification of SMRA is given in [7]. A router i can be active or passive for destination j at any given time. Node i is active for destination j if it is waiting for at least one reply from a neighbor, and is passive otherwise. A router i initializes itself in passive state with an infinite distance to all its known neighbors and a zero distance to itself. The maximum allowable distance to reach neighbor, defined below, is also set to ∞. Routers send updates containing distance and credit information for themselves to all their neighbors. When the destinations become operational, routers inform their neighbors about the available credits to all other nodes.

Credit information is updated periodically while the distance information is exchanged among neighbors when the state of the network changes. Each routing update updates the cost and the credit information. An update can be a full routing table or increments of the routing table in different update messages. After initialization, only incremental updates are sent.

For a given destination, a router updates its routing table differently depending on whether it is *passive* or *active* for that destination. A router that is passive for a given destination can update the routing-table entry for that destination independently of any other routers, and simply chooses as its new distance to the destination to be the shortest distance to that destination among all neighbors, and as its new feasible successor to that destination to be any neighbor through whom the shortest distance is achieved. In contrast, a router that is or becomes active for a given destination must synchronize the updating of its routing-table entry with other routers.

When a router is passive and needs to update its routing table for a given destination j after it processes an update message from a neighbor or detects a change in the cost or availability of a link or a change in the credit information, it tries to obtain a *feasible successor*. From router i's standpoint, a feasible successor toward destination j is a neighbor router k that satisfies the maximum allowable distance condition (MADC) given by the following two equations [5]:

$$\begin{aligned} D^i_j &= D^i_{jk} + d_{ik} = Min\{D^i_{jp} + d_{ip} \mid p \in N_i\} \\ D^i_{jk} &< MAD^i_j \end{aligned} \tag{13}$$

where MAD^i_j is the *maximum allowable distance* for destination j, and is equal to the minimum value obtained for D^i_j since the last time router i transitioned from active to passive state for destination j. Router i adjusts MAD^i_j depending on the congestion level of the network.

If router i finds a feasible successor, it remains passive and updates its routing-table entry as in the Distributed Bellman-Ford algorithm [2]. Alternatively, if router i cannot find a feasible successor, it first sets its distance equal to the addition of the distance reported by its current successor plus the cost of the link to that neighbor. The router also sets its maximum allowable distance equal to its new distance. After performing these updates, the router becomes active by sending a query in an update message to all its neighbors; such a query specifies the router's new distance through its current successor. It then sets the destination's reply-status table entry for each link to one, indicating that it expects a

reply from each neighbor for that destination.

Once active for destination j, router i cannot change its feasible successor, MAD_j^i, the value of the distance it reports to its neighbors, or its entry in the routing table, until it receives all the replies to its query. A reply received from a neighbor indicates that such a neighbor has processed the query and has either obtained a feasible successor to the destination, or determined that it cannot reach the destination. Once node i obtains all the replies to its query, it computes a new distance and successor to destination j, updates its feasible distance to equal its new distance, and sends an update to all its neighbors.

Multiple changes in link cost or availability are handled by ensuring that a given node is waiting to complete the processing of at most one query at any given time. The mechanism used to accomplish this is specified in [5], and is such that a node can be either passive or in one of four active states, and it processes any pending update or distance increases that occurred while it was active. The state of node i for destination j is denoted by the flag o_j^i.

Ensuring that updates stop being sent in the network when some destination is unreachable is easily done. If node i has set $D_j^i = \infty$ already and receives an input event (a change in cost or status of link $(i,\ k)$, or an update or query from node k) such that $D_{jk}^i + d_{ik} = \infty$, then node i simply updates D_{jk}^i or d_{ik}, and sends a reply to node k with $RD_j^i = \infty$ if the input event is a query from node k. When an active node i has an infinite maximum allowable distance and receives all the replies to its query such that every neighbor offers an infinite distance to the destination, the node simply becomes passive with an infinite distance.

When node i establishes a link with a neighbor k, it updates the value of d_{ik} and assumes that node k has reported infinite distances to all destinations and has replied to any query for which node i is active. Furthermore, if node k is a previously unknown destination, node i sets $o_k^i = 1$, $s_k^i = null$, and $D_k^i = RD_k^i = MAD_k^i = \infty$. Node i also sends to its new neighbor k an update for each destination for which it has a finite distance.

When node i is passive and detects that link $(i,\ k)$ has failed, it sets $d_{ik} = \infty$ and $D_{jk}^i = \infty$. After that, node i carries out the same steps used for the reception of a link-cost change in the passive state.

Because a router can become active in only one diffusing computation per destination at a time, it can expect at most one reply from each neighbor. Accordingly, when an active node i loses connectivity with a neighbor n, node i can set $r_{jn}^i = 0$ and $D_{jn}^i = \infty$, i.e., assume that its neighbor n has sent any required reply reporting an infinite distance. If node n is s_j^i, node i also sets $o_j^i = 0$. When node i becomes passive again and $o_j^i = 0$, it cannot simply choose a shortest distance; rather, it must find a neighbor that satisfies the MADC using the value of MAD_j^i set at the time node i became active in the first place. After finding a new successor, the permit bucket parameters ρ_j^i and σ_j^i are also updated.

Figure 5 gives a graphical representation of how MAD is updated. The point at which a new diffusing computation starts is a *synchronization point*. It can be noted that between two synchronization points the value of MAD can only decrease or remain the same.

To route packets to a destination j, each router uses the following rule to select the neighbor routers that should belong to its shortest multipaths for j:

Shortest Multipath Condition (SMC): At time t, router i can make node $k \in N_i(t)$ part of SM_j^i if and only if $D_{jk}^i(t) < MAD_j^i(t)$.

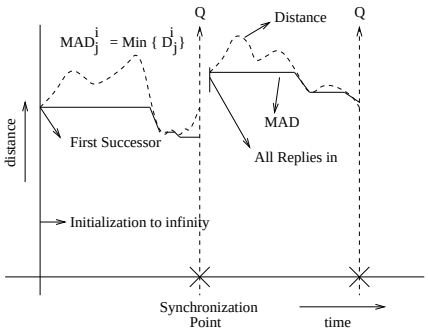

Fig. 5. Maximum Allowable Distance Condition

When nodes choose their successors using SMC, the path from source to destination obtained as a result of this is loop free at every instant. The proof of correctness and loop-freedom of SMRA is basically the same as that provided in [5] for DUAL.

4. Worst-Case Steady-State Delays

In this section, we derive an upper bound on the end-to-end steady-state path delay from node i to destination j (D_j^{i*}) as a function of the credits available through each path under steady state. Steady-state means that all distances and credit information is correct at every router. This bound demonstrates that it is possible to provide performance guarantees in a connectionless routing architecture. The delay experienced by a packet accepted into the network is the time required by a data packet to reach its destination router from a source. This includes both the propagation delay and the queueing delay. Path delay can also be interpreted as the time it would take for a destination j backlog to clear when there are no more arrivals after time t.

Parekh and Gallager have analyzed worst-case session delay in a connection-oriented network architecture [10]. We adopt a similar approach for each destination in a connectionless architecture. To do this, we assume a stable topology in which all routers have finite distances to each other. We also make use of the fact that SMRA enforces loop-freedom at every instant on all paths in the shortest multipath sets.

In a connectionless network where routes are computed distributedly, the path taken by a packet can change dynamically depending on the congestion level in the network. Routing is done on a hop-by-hop basis, independently at each router. Therefore, the total traffic at a node will be the sum of the traffic on all its links connecting to upstream neighbors. To obtain an expression for the worst-case bound, we make the following assumptions:

1. Each node sends traffic to destination j as long as credits are available (non-zero) for that destination along any of its chosen paths.
2. At every node m, traffic for every destination is treated independently.
3. Traffic arriving at a node i for destination j in the interval $(0, t)$ (denoted by A_j^i) is the sum of the traffic from all its upstream neighbors to destination j and the traffic originated at the node i itself, denoted by $r_j^i(t)$, i.e.,

$$A_j^i(t) = r_j^i(t) + \sum_{n, i \in SM_j^n(t)} f_{ji}^n(t) \quad (14)$$

$$= r_j^i(t) + \sum_{l | i \in SM_j^l(t)} \phi_{jl}^i(t).A_{jl}^i(t) \quad (15)$$

Each router in a connectionless network can itself be a source to any given destination. At each node, traffic to destination j is constrained by a permit bucket filter. The worst-case delay and backlog is upper bounded by an additive scheme due to Cruz [3]. The rate at which the packets are serviced at each node depends on the permit bucket or leaky bucket parameters σ_j^i and ρ_j^i for a given destination j. The parameter σ_j^i gives the permit bucket size and ρ_j^i the credit generation rate at node i. Therefore, the number of packets that are being serviced at a node is a function of σ_j^i and ρ_j^i.

The minimum service rate g_{jm}^i at any node i is the fraction of the input traffic at node i for destination j . The fraction of the traffic is determined by the ratio of the routing variables of the links, which is a function of the traffic flow; Therefore,

$$g_{jm}^i = \frac{\phi_{jm}^i}{\sum_{k=1}^{SM_j^i(t)} \phi_{jk}^i} \, t_j^i \tag{16}$$

The minimum clearing rate of a given path is $g_l^{'} = \min_{m \in P(i,j)} \; g_{jm}^i$. When $g_l^{'} > \rho_j^l$, the system with respect to destination j is said to be locally stable. The input traffic rate at node i to destination j is the sum of all the incoming traffic destined for j for which i is the intermediate node and the traffic originated at i itself (Eq. 5). With these constraints, the bound on the delay for a given destination can be obtained using a similar approach as in [10].

The delay on a link (i,k) (per hop delay) d_{jk}^i for a given destination j is the sum of the queueing delay and the propagation delay on that link. The link propagation delay (δ_k^i) depends on the congestion level of the link as well as the link capacity. Propagation delay is defined as the time taken for a packet to reach a destination from a source. Every packet is time-stamped when it leaves a node and the time at which the packet reaches the neighbor is noted. The difference between the two gives a one-hop delay. The average of this delay over a given period of time gives the propagation delay δ_k^i.

The queueing delay is the time a packet has to wait at a node before it is processed. The waiting time of a packet depends on the number of packets already present in the queue at the time a packet arrives. This is referred to as the *backlog* at node i for destination j and is denoted by Q_j^i. Therefore, the delay on link (i,l) for destination j at time t is

$$d_{jl}^i(t) = \delta_l^i(t) + Q_{jl}^i(t).\delta_l^i(t) = \delta_j^i(t)[1 + Q_{jl}^i(t)] \tag{17}$$

The backlog number of packets for a given destination j at a given time t can be defined as the difference in the incoming and the outgoing traffic at a node, i.e.,

$$Q_j^i(t) = A_j^i(t) - S_j^i(t) \tag{18}$$

This takes into account both the processing delay and the queueing delay experienced at each hop. For every interval $(\tau, t]$,

$$S_j^i(\tau, t) \geq (t - \tau) g_i^{'} \tag{19}$$

If the minimum clearing time $g_i^{'}$ is greater than the token generation rate ρ_j^i for a given destination, we can obtain a bound on the backlog and hence the path delay. Let $\tau < t$ be the time at which there are no backlogged packets in the network. Then, because $g_j^i \geq \rho_j^i$ and all the destinations are permit bucket constrained,

$$S_j^i(\tau, t) \geq (t - \tau)\rho_j^i(t - \tau) \tag{20}$$

4.1 Negligible Packet Size

We first obtain a bound on end-to-end path delay assuming that the size of the packet may not contribute significantly to the delay component. The arrivals at each node i is the sum of the arrivals at all the upstream nodes for destination j and the traffic originated at node i itself. For all $t \geq \tau \geq 0$ we have,

$$A_j^i(\tau, t) = r_j^i(\tau, t) + \sum_{l | i \in SM_j^l(t)} \phi_{jl}^i(\tau, t).A_j^l(\tau, t) \tag{21}$$

The maximum backlog traffic Q_j^{i*} for destination j is the difference between the arrivals in the interval $(\tau, t]$ and the total packets serviced in the same interval at node i. For $\phi_{jl}^i > 0$,

$$\begin{aligned}
Q_j^{i*}(\tau, t) &\leq A_j^i(\tau, t) - S_j^i(\tau, t) && (22)\\
Q_j^{i*}(\tau, t) &\leq r_j^i(\tau, t) - S_j^i(\tau, t) \\
&\quad + \sum_{l | i \in SM_j^l(t)} [\phi_{jl}^i(\tau, t).A_j^l(\tau, t)] && (23)\\
Q_j^{i*}(\tau, t) &\leq [r_j^i(t) - r_j^i(\tau)] - S_j^i(\tau, t) \\
&\quad + \sum_{l | i \in SM_j^l(t)} [\phi_{jl}^i(\tau, t).A_j^l(\tau, t)] && (24)
\end{aligned}$$

The difference $(r_j^i(t) - r_j^i(\tau))$ determines the amount of traffic arriving at node i in the interval $(t - \tau)$; the maximum of which is the sum of the tokens available at node i and the tokens received in the interval $(t - \tau)$. At every node, each destination is constrained independently by a permit bucket scheme. Following Parekh and Gallager's approximation [10], we assume the links to be of infinite capacity. The results for the infinite capacity case upper-bound the finite capacity case. In other words, the results of infinite capacity can be used for any finite speed link. The arrival and the service functions at each router can be translated to permit bucket parameters which in turn depend on the maximum tolerable path delay and the link flows. Substituting for the arrivals and the number of packets serviced in terms of the permit bucket parameters from the previous section we have

$$\begin{aligned}
Q_j^{i*}(\tau, t) &\leq [\sigma_j^i(t - \tau) + \rho_j^i(t - \tau)] - \rho_j^i(t - \tau) \\
&\quad + \sum_{l | i \in SM_j^l(t)} \frac{f_{jl}^i}{t_j^i} [\sigma_j^i(t - \tau) + \rho_j^i(t - \tau)]
\end{aligned}$$

Because $\frac{f_{jl}^i}{t_j^i} \leq 1$ for any j and $l \in SM_j^i(t)$,

$$Q_j^{i*}(\tau, t) \leq \sigma_j^i(t - \tau) + \sum_{l \; | \; i \in SM_j^l(t)} [\sigma_j^l(t - \tau) + \rho_j^l(t - \tau)] \tag{25}$$

Making $\tau = t - \Delta t$, we can write

$$Q_j^{i*}(t) \leq \sigma_j^i(t) + \sum_{l \; | \; i \in SM_j^l(t)} [\sigma_j^l(t) + \rho_j^l(t)]$$

Therefore, the backlog at node i to destination j depends on the leaky bucket parameters at node i and the permit bucket parameters of all the upstream neighbors of i for which node i is in the shortest multipath set.

The delay at each node i can be computed as the weighted average path delay through all its multipath neighbors; therefore,

$$D_j^i(t) = \sum_{k \in SM_j^i(t)} \phi_{jk}^i(t) \check{D}_{jk}^i(t) \quad (26)$$

The distance from i to j through neighbor k can be expressed as the sum of the distance from k to j and the link cost from i to k. The link cost is the sum of the distance and the propagation delay of that link. Therefore,

$$\begin{aligned} \check{D}_{jk}^i(t) &= D_{jk}^i(t) + d_{ik}(t) = D_{jk}^i(t) + [d_{jk}^i(t) + \delta_k^i(t)] \\ D_j^i(t) &= \sum_{k \in SM_j^i(t)} \phi_{jk}^i(t)[D_{jk}^i(t) + (d_k^i(t) + \delta_k^i)(t)] \end{aligned} \quad (27)$$

From Eq. 17, $d_{ik}(t) = \delta_k^i(t)[1 + Q_j^i(t)]$, which implies that

$$D_j^i(t) = \sum_{k \in SM_j^i(t)} \phi_{jk}^i(t)[D_{jk}^i(t) + \delta_k^i(t)(1 + Q_j^i(t))] \quad (28)$$

Because SMC must be satisfied by every $k \in SM_j^i(t)$, $D_{jk}^i(t) < MAD_j^i(t)$. Then, if $D_j^i(\tau)$ is the maximum path delay from i to j at time τ and $Q_j^{i*}(t)$ is the maximum backlog from i to j at time t, we obtain from Eq. 28 that

$$\begin{aligned} D_j^{i*}(t) &< \sum_{k \in SM_j^i(t)} [\phi_{jk}^i(t).\delta_k^i(t)(1 + Q_j^{i*})(t)] \\ &\quad + MAD_j^i(t) \sum_{k \in SM_j^i(t)} \phi_{jk}^i(t) \end{aligned} \quad (29)$$

Let the maximum link propagation delay of all the links from i to a node in $SM_j^i(t)$ be

$$\Delta_j^i(t) = \max_{k \in SM_j^i(t)} \delta_k^i(t) \quad (30)$$

Therefore, the maximum path delay from i to j becomes

$$\begin{aligned} D_j^{i*}(t) &< \Delta_j^i(t) \sum_{k \in SM_j^i(t)} \phi_{jk}^i(t)[1 + Q_j^{i*}(t)] \\ &\quad + MAD_j^i(t) \sum_{k \in SM_j^i(t)} \phi_{jk}^i(t) \end{aligned} \quad (31)$$

Noticing that $Q_j^{i*}(t)$ is independent of k and substituting Eq. 7 in Eq. 31 we obtain

$$D_j^{i*}(t) < \Delta_j^i(t)[1 + Q_j^{i*}(t)] + MAD_j^i(t) \quad (32)$$

The above equation is an upper bound on $D_j^i(t)$ that should be expected. It states that $D_j^i(t)$ must be smaller than the sum of the product of the backlog for i at node i times the maximum link propagation delay in node i's shortest multipath, plus $MAD_j^i(t)$. The first term of Eq. 32 corresponds to the delay incurred by sending all backlogged packets at time t to a neighbor with the longest link propagation delay. The second term corresponds to the maximum delay incurred by any neighbor receiving the backlog packets; because any such neighbor must be on $SM_j^i(t)$, that delay can be at most equal to $MAD_j^i(t)$.

Substituting Eq. 25 in Eq. 32, we can represent the same bound in terms of permit bucket parameters as follows:

$$\begin{aligned} D_j^{i*}(t) &< \Delta_j^i(t)\{1 + \sigma_j^i(t) \sum_{l | i \in SM_j^l(t)} [\sigma_j^l(t) + \rho_j^l(t - \tau)]\} \\ &\quad + MAD_j^i(t) \end{aligned} \quad (33)$$

The bound given by Eqns. 32 and 33 for router i is based on a maximum delay offered by the neighbor of i and a maximum backlog allowed at router i. This is possible because of two main features of SMRA: datagrams are accepted only if routers have enough credits to ensure their delivery, and datagrams are delivered along loop-free paths. In contrast, in traditional datagram routing architectures, any datagram presented to a router is sent towards the destination, and the paths taken by such datagrams can have loops; therefore, it is not possible to ensure a finite delay for the entry router or any relay router servicing a datagram.

4.2 Non-negligible Packet Size

In PGPS networks, routing nodes do not transmit packets until a packet has completely arrived. Therefore, the number of packets which will reach a downstream node is at the most equal to the number of packets serviced by its upstream neighbors. Let L_i be the maximum packet size at node i. The PGPS server does not begin servicing a packet until the last bit has arrived.
For a packet-switched network

$$A_j^i(\tau, t) = r_j^i(\tau, t) + \sum_{m \mid i \in SM_j^m(t)} S_{ji}^m(\tau, t) \quad (34)$$

Here, $S_{ji}^m(\tau, t)$ represents the number of packets serviced by an upstream neighbor m for which i is in the shortest multipath to j in the interval $(t - \tau)$. Let K be the number of hops in a given path from i to j; m and $m - 1$ be two successive nodes. Then, for a given path,

$$\begin{aligned} \sum_{m-1} S_{jm}^{m-1}(\tau, t) &\geq A_j^m(\tau, t) - r_j^m(\tau, t) \\ &\geq \sum_{m-1} [S_{jm}^{m-1}(\tau, t) - L_{m-1} \end{aligned} \quad (35)$$

where, $m = 2, ..., K, \tau < t$ and L_{m-1} is the maximum length of a packet transmitted by node $m - 1$. Here, the nodes m and $(m - 1)$ are such that $m \in SM_j^{m-1}$.

For a PGPS system, the number of packets serviced for $t > \tau$ is given as

$$\begin{aligned} S_j^i(\tau, t) &\geq \min_{V \in [\tau, t]} \{[A_j^i(\tau, V) - r_j^i(\tau, V)] \\ &\quad + G_j^K(t - V)\} + K.L_i \end{aligned} \quad (36)$$

where V represents the last time in the interval $[\tau, t]$ at which node i begins a busy period for destination j and the function G_j^K is a convex function which indicates the amount of service given to destination j under a greedy regime.

$$S_j^i(0,t) \geq \min_{V\in[0,t]}\{[A_j^i(0,V) - r_j^i(0,V)] + G_j^K(t-V)\} + K.L_i \quad (37)$$

With a greedy regime, the service to destination j is minimized and is delayed by an appropriate amount, which is given by the minimizing value of V, denoted by V_{min}.

$$S_j^i(t) \geq \{[A_j^i(V_{min}) - r_j^i(V_{min})] + G_j^K(t - V_{min})\} + K.L_i \quad (38)$$

The backlog traffic for a destination j from i is the difference between the number of packets that has arrived and the number of packets serviced as in the previous case.

$$Q_j^i(0,t) = A_j^i(0,t) - S_j^i(0,t) \quad (39)$$

Applying similar argument as in the previous section, we have

$$Q_j^i(t) = \sigma_j^i(t) + \rho_j^i(t) + \sum_{l|SM_j^i(t)\in l} [\sigma_j^l(t) + \rho_j^l(t)] - S_j^i(t) \quad (40)$$

Substituting for $S_j^i(t)$ we have,

$$\begin{aligned} Q_j^i(t) &= \rho_j^i(t) - \rho_j^i(V_{min}) - G_j^m(t - V_{min}) + m.L_i \\ &\quad + \sum_{l|SM_j^i(t)\in l} [\sigma_j^l(t) + \rho_j^l(t)] \end{aligned} \quad (41)$$

Thus, the maximum backlog is given by

$$\begin{aligned} Q_j^{i*}(t) &= \rho_j^i(t) - \rho_j^i(V_{min}) + m.L_{max} - G_j^m(t - V_{min}) \\ &\quad + \sum_{l|SM_j^i(t)\in l} [\sigma_j^l(t) + \rho_j^l(t)] \end{aligned} \quad (42)$$

Having bound the worst-case backlog, we can use a similar approach as in Eq. 31 to obtain bounds for the maximum path delay. Since we are considering a PGPS system, the expression for maximum path delay becomes

$$D_j^{i*}(t) \leq MAD_j^i(t) + \Delta_j^i(t) \sum_{l|i\in SM_j^l(t)} [1 + Q_j^{i*}(t)] \quad (43)$$

Substituting for the maximum backlog from Eq. 42 we obtain,

$$\begin{aligned} D_j^{i*}(t) &\leq MAD_j^i(t) + \Delta_j^i(t) \sum_{l|i\in SM_j^l(t)} \{1 + \rho_j^i(t) - \rho_j^i(V_{min}) \\ &\quad + m.L_{max} - G_j^m(t - V_{min}) \\ &\quad + \sum [\sigma_j^l(t) + \rho_j^l(t)]\} \end{aligned} \quad (44)$$

$$\begin{aligned} D_j^{i*} &\leq MAD_j^i(t) + \Delta_j^i(t)\{1 + \rho_j^i(t) - \rho_j^i(V_{min}) \\ &\quad + m.L_{max} - G_j^m(t - V_{min}) \\ &\quad + \sum_{l|i\in SM_j^l(t)} [\sigma_j^l(t) + \rho_j^l(t)]\} \end{aligned} \quad (45)$$

Here again, the excess delay experienced by a packet depends on the network traffic as earlier. In addition, it also is a function of the packet size and depends on the entire path from source to a given destination node.

5. Conclusion

We have presented a new framework for the modeling of multipath routing in connectionless networks that dynamically adapt to network congestion. We have demonstrated that it is possible to provide performance guarantees for the delivery of packets in such networks. The basic routing protocol uses a short-term metric based on hop-by-hop credits to reduce congestion and a long term metric based on end-to-end path delays from a source to a destination. Packet forwarding is done on a hop-by-hop basis. Each node is modeled as a PGPS server which contains destination-based permit buckets. A loop-free multipath routing protocol has been proposed to regulate the traffic on each link by monitoring the parameters of the destination-oriented permit buckets. A worst-case delay bound under steady-state has been derived for the above network model for both negligible and non-negligible packet sizes.

Our work continues to study the dynamic behavior of congestion-oriented shortest multipath routing, and to define how destination-oriented routing mechanisms can be used to satisfy performance requirements specified by the sources of packets.

References

[1] D. Bertsekas. Dynamic behavior of shortest path routing algorithms for communication networks. *IEEE Trans. Automat. Control*, AC-27, pp.60–74, 1982.

[2] D.P. Bertsekas and R.G. Gallager. *Data Networks*. Prentice Hall, 1992.

[3] R.L. Cruz. A calculus for network delay, Part II: Network Analysis. *IEEE Trans. Inform. Theory*, Vol.37, pp.132–141, 1991.

[4] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM*, pp.1–12, 1989.

[5] J.J. Garcia-Luna-Aceves. Loop-free routing using diffusing computation. *IEEE/ACM Trans. Networking*, Vol.1, No.1, pp.130–141, Feb 1993.

[6] H.T Kung, Blackwell. T, and Chapman. A. Credit-based flow control for ATM networks: credit update protocol, adaptive credit allocation, and statistical multiplexing. In *ACM SIGCOMM*, pp.101–14, London, UK, Aug/Sept 1994.

[7] Shree Murthy and J.J. Garcia-Luna-Aceves. Congestion-oriented shortest multipath routing. UCSC-Technical Report, Oct. 1995.

[8] C. Ozveren, R. Simcoe, and G. Varghese. Reliable and efficient hop-by-hop flow control. In *ACM SIGCOMM*, pp.89–100, 1994.

[9] A.K. Parekh and R.G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. *IEEE/ACM Trans. Networking*, Vol.1, No.3, pp.344–357, June 1993.

[10] A.K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. *IEEE/ACM Trans. Networking*, Vol.2, No.2, pp.137–150, April 1994.